Distributed Logging

Suresh Podeti

4 min read · Sep 27, 2023

Photo by Boba Jaglicic on Unsplash

Introduction

A **log file** records details of events occurring in a software application. The details may consist of microservices, transactions, service actions, or anything helpful to debug the flow of an event in the system. Logging is crucial to monitor the application's flow.

Logging is essential in **understanding the flow of an event** in a distributed system. It seems like a tedious task, but upon facing a failure or a security breach, logging helps pinpoint when and how the system failed or was compromised. It can also aid in finding out the root cause of the failure or breach. It decreases the meantime to repair a system.

Why don't we simply print out our statements to understand the application flow?

- The output of print functions usually goes to the terminal, while our need could be to **persist** such data on a local or remote store.
- Moreover, we can have millions of print statements, so it's better to structure and store them properly.
- Concurrent activity by a service running on many nodes might need causality information to stitch together a correct flow of events properly

Logging allows us to understand our code, locate unforeseen errors, fix the identified errors, and visualize the application's performance.

Logging in a distributed system

In today's world, an increasing number of designs are **moving to microservice architecture instead of monolithic architecture**. In microservice architecture, logs of each microservice are accumulated in the respective machine. If we want to know about a certain event that was processed by several microservices, it is difficult to go into every node, figure out the flow, and view error messages. But, it becomes handy if we can trace the log for any particular flow from end to end.

Restrain the log size

The number of logs increases over time. At a time, perhaps hundreds of concurrent messages need to be logged. But the question is, are they all important enough to be logged? To solve this, logs have to be structured. We need to decide what to log into the system on the application or logging level.

Use sampling

We'll determine which messages we should log into the system in this approach. Consider a situation where we have lots of messages from the same set of events.Instead of logging all the information, we can use a **sampler service** that only logs a smaller set of messages from a larger chunk. This way, we can decide on the most important messages to be logged.

Use categorization

We can also categorize the types of messages and apply a filter that identifies the important messages and only logs them to the system.

The following severity levels are commonly used in logging:

- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`
- `FATAL/CRITICAL`

Requirements

Let's list the requirements for designing a distributed logging system:

- **Low latency**: Logging is an I/O-intensive operation that is often much slower than CPU operations. We need to design the system so that logging is not on an application's critical path.
- **Scalability:** We want our logging system to be scalable. It should be able to handle the increasing amounts of logs over time and a growing number of concurrent users.
- **Availability:** The logging system should be highly available to log the data.

Design

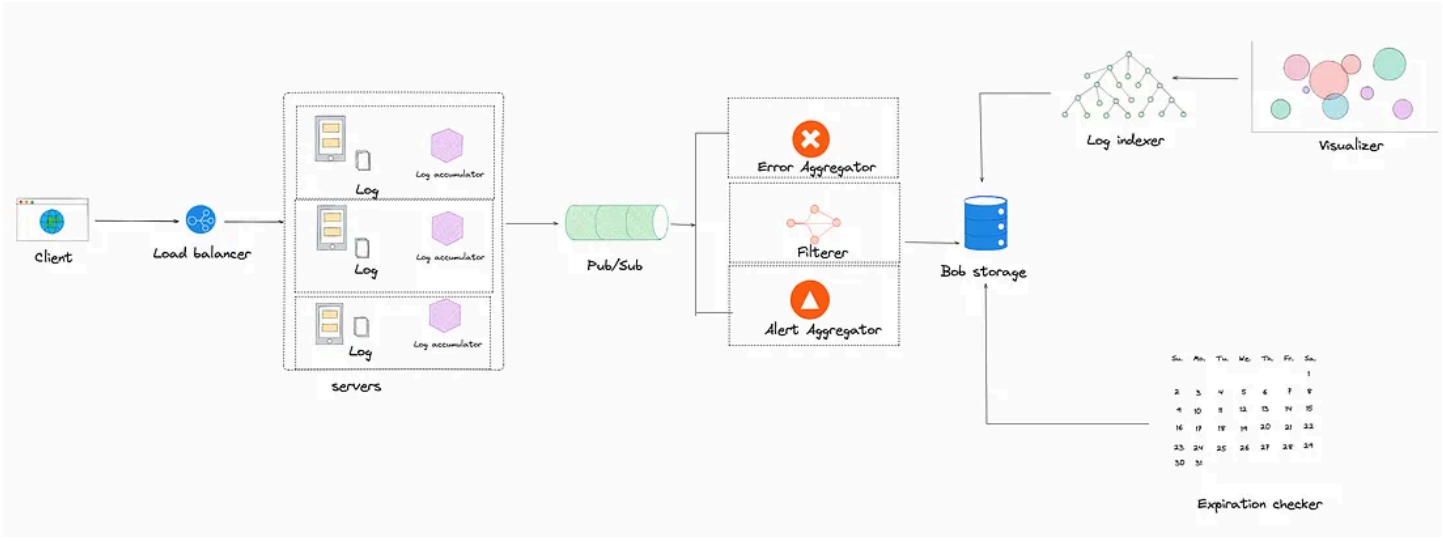

Fig 1.0: High Levl Design of distributed logging system

Let's list the major components of our system:

- **Log accumulator**: Each server has its log accumulator, and does three actions receives logs, storing them locally, and push the logs to pub/sub system.Let's consider a situation where we have multiple different applications on a server, such as App 1, App 2, and so on. Each application has various microservices running as well. We use an ID with `application-id`, `service-id`, and its time stamp to uniquely identify various services of multiple applications.
- **Pub/Sub**:All servers in a data center push the logs to a pub-sub system. Since we use a horizontally-scalable pub-sub system, it is possible to manage huge amounts of logs. We may use multiple instances of the pub-sub per data center. It makes our system scalable, and we can avoid bottlenecks. Then, the pub-sub system pushes the data to the blob storage
- **Filterer:** It identifies the application and stores the logs in the blob storage reserved for that application since **we do not want to mix logs** of two different applications
- **Error aggregator**: It is critical to identify an error as quickly as possible. We use a service that picks up the error messages from the pub-sub system and informs the respective client. It saves us the trouble of searching the logs.
- **Alert aggregator**: Alerts are also crucial. So, it is important to be aware of them early. This service identifies the alerts and notifies the appropriate stakeholders if a fatal error is encountered, or sends a message to a monitoring tool.
- **Blob storage**: The logs need to be stored somewhere after accumulation. We'll choose blob storage to save our logs.
- **Log indexer**: The growing number of log files affects the searching ability. The log indexer will use the distributed search to search efficiently.
- **Visualizer**: The visualizer is used to provide a unified view of all the logs.
- **Expiration checker:** Verifying the logs that have to be deleted. Verifying the logs to store in cold storage.

Distributed Log | Distributed System Design | Software Architecture

Software Architect Logging And Monitoring

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium

 Santanu Sahoo in Booking.com Engineering